

.NET Industrial Automation

Steven L. Weygandt, PE – Invensys Production Management
Dave Hardin, PE, CSDP – Invensys Production Management

Keywords

Industrial automation, software, Windows, .NET framework, C#

Abstract

.NET is a Microsoft networking and computing framework. .NET technology can be applied to the creation of software solutions for real-time industrial automation systems. Spanning from embedded control in devices all of the way up to system integration with enterprise business systems there are productivity, connectivity, scalability, and reliability issues. This paper describes specific .NET software technologies and tools, their capabilities and their limitations. They are described in the context of real world, real-time industrial automation.

Introduction

Industrial automation applications have been implemented using a variety of software tool sets directed at a number of different operating systems with varying degrees of commercial success. Any operating system can theoretically be applied to industrial automation and in fact many of them have been. Unchecked variety, however, has significantly increased automation system entropy (a term borrowed from physics.) Herein it refers to a measure of the complexity of software interfaces in industrial systems. Complex, piece-meal, high-entropy automation systems, when analyzed for life-cycle economics, are expensive.

This technical paper focuses on Microsoft's .NET framework running on Windows XP, CE, and the planned .NET Server. It presumes development with programming languages supported in the Visual Studio .NET environment. It elaborates on these topics given three premises.

Premise 1) Control automation relies upon software to do the following:

- link field sensors, transmitters, actuators, etc.
- execute loop control, sequence control, hybrid control, etc.
- implement cell control, supervisory control, unit optimization, etc.
- inter-operate with facility management systems, laboratory information systems, etc.
- integrate with continuous process and/or discrete manufacturing business systems
- other automation tasks.

Industrial automation linking, executing, implementing, inter-operating, and integrating can be addressed using .NET as an application framework and by leveraging its standards-based information exchange technologies. Caveats regarding what needs to be done with .NET to insure industrial performance are covered in this paper.

Premise 2) Selection of processor hardware and operating system software for control automation components may be factored according the following categories of requirements:

- signal interface (electrical, optical, electro-magnetic)
- frequency (of signal sample and of loop execution)
- robustness and reliability (hardware and software)
- security (physical and programmatic)
- tool set (data structures and algorithms)

Industrial automation solutions are composites of software modules running on hardware components. Each module needs to effectively address some aspect of the overall automation solution while inter-operating with other modules. In choosing the software tool set, selection factors cited above are heavily weighted by the degree to which components and tools fit automation requirements. The .NET framework delivers a broad base of services covering a broad range of functionality. Visual Studio .NET languages are capable of providing realistic abstractions of automation functions. What this means is that the objects programmed using source code of selected .NET programming languages can be clearly mapped to specific real aspects of automation objects. In particular the C# language introduced with .NET promotes elimination of intervening layers of abstraction in software. The concept of software automation objects modeled from real world processes will be further detailed in this paper.

Premise 3) All production facilities must, to some extent, implement automation in order to be commercially successful. Ubiquitous industrial automation may be applied for new or retrofit production automation systems.

Ubiquitous industrial automation is a software application endeavor driven by software tool sets designed for select operating systems. The goal is to develop a set of multi-purpose, re-useable, robust, reliable, yet secure software objects that are adaptable to many automation scenarios. An enterprise driven economic requirement is to reduce the cost of automation through rapid development, quicker configuration and effortless deployment of automation functionality. Tools are needed that effectively enhance productivity for each of these tasks. Ubiquitous software objects address productivity from application development to deployment. .NET can play a significant role in facilitating improvements in productivity associated with the creation of ubiquitous software objects.

This premise applies to continuous process facilities, discrete manufacturing facilities and hybrids. Ubiquitous industrial automation also will be a major factor in achieving cost effective integration of facilities, both intra-enterprise and inter-enterprise. There is an innate desire to keep enterprise system acquisition costs and long-term operating costs in check. This means acquiring and maintaining effective yet relatively inexpensive automation components. Achieving a rich set of ubiquitous industrial automation components would assure low acquisition cost because volume pricing models come into play. A rich set of broadly available components also assures competition for the delivery of development and maintenance services, leading to lower service charges and significantly lower life-cycle cost.

Disclaimer About Acronyms and Abbreviations (DAAA)

Acronyms and abbreviations are fundamental to a thorough discussion about software-based control systems. Important abbreviations and acronyms are defined upon first reference and listed in the glossary at the end of this paper.

What is the CLR?

The acronym 'CLR' shows up everywhere in software periodicals and books that have anything to do with Microsoft. Its definition will be dealt with first. CLR represents Common Language Runtime, three words that describe a new approach to the execution of software code in a Microsoft operating system environment. 'Common Language' refers to Microsoft's support for several standard programming languages including C# (pronounced 'see sharp'), VB.NET, C++.NET, and J#.NET. A number of non-Microsoft languages also generate code for execution in the CLR. The 'R' in CLR refers to 'runtime' execution of compiled code. Execution is implemented by EXEs and supported by DLLs (dynamic link libraries) for a wide variety of object classes in the .NET Framework. DLLs in .NET are related in purpose to those compiled from C language, C++ or even Visual Basic yet .NET versions provide much more autonomous functionality. The .NET 'framework' is marketing terminology for all of the stuff that goes together to make software applications work within and connecting to the new .NET environment.

Common Language Infrastructure (CLI) is a designation for documents that contain the specification of components, structures and rules for much of the .NET framework. Microsoft has magnanimously exposed much of the details of the infrastructure in a source code distribution (called 'Rotor'). This source code is available for download and can be compiled for purposes of non-commercial evaluation and instruction. The important opportunity regarding this source code is that inspection of it by experienced programmers reveals a lot about the intimate details of how the CLR and other components of .NET work.

The code that gets executed within the CLR is in fact MSIL (Microsoft Intermediate Language) – IL for short. IL is akin to Java Byte Code. In fact, many of the concepts in the .NET CLR have been proven in Java software. In particular the C# language is built with software object modeling as a fundamental design tenet, much like Java. Source code in any of the .NET languages, for example C#, is compiled. Typically this is done using Visual Studio.NET. Compilation yields 'assemblies' of IL code. Assemblies are installed and executed as 'managed' code using services of the CLR.

IL code is translated for runtime execution on a processor by a tool called the JIT (just-in-time) compiler which efficiently builds the correct machine code for the target operating system/processor just before it is needed. The CLR provides a mechanism that determines when parts of IL encoded assemblies need to be translated to machine byte code. It caches the translated code for re-use. A disassembler program may be used to inspect assembly code modules and there is a set of debugging tools available with Visual Studio.NET that supports multiple languages simultaneously. Assemblies running on the CLR may in fact be compiled from different source languages and unified within a common package with high expectations that they will all inter-operate.

Assemblies, the Global Assembly Cache (GAC) and Side-by-Side

The notion of assemblies running in the .NET CLR has provided a nearly complete answer to the problem of conflicts in past Windows software applications between differing versions of Win32 DLLs. In the name of progress, operating systems and programming languages plus supporting libraries of functions have rapidly evolved. Windows in particular has kept up the pace. .NET upon its official release is actually not at the beginning of its evolutionary cycle. More than four years of development and testing went into the .NET framework, the new language (C#), the new versions of other languages (VB.NET, C++.NET, J#.NET), and a system of supporting services built which are deployed as a set of framework classes.

One of the biggest problems that needed to be tackled for reliability of software was the issue of keeping up with changes in support DLLs. What has been the problem with support DLLs on Windows? Microsoft Foundation Classes (MFCs) and COM libraries have been notorious for introducing frequent changes that support 'new and improved' features. These updates result in the release of new DLLs. Even when a module's feature set was complete, the DLLs were changed anyway to fix bugs. MFC and COM software have proliferated throughout the world of industrial software automation. It is always an intense exercise in frustration attempting to sort out the correct DLL versions on computers that need to implement diverse MFC and COM automation components. .NET has addressed this problem utilizing a technology called 'managed assemblies'.

Assemblies of various modules compiled from C#, VB.NET and C++.NET are organized for simple deployment. An assembly contains code and data structures which contain both data and metadata. Metadata is a way of embedding information about the parts and pieces of the assembly. Metadata also describes the dependencies of the modules, for example defining a particular version of a library or a directory location for finding a module that another depends upon. A self-contained assembly, or the primary assembly of a set of interdependent assemblies contains a special kind of metadata called the 'manifest'. The manifest adheres to a precise specification that enforces a complete description of everything that a software program does, contains, or needs for its functionality.

Windows registration (i.e. the Windows Registry) is not required for .NET assemblies. Registry keys are supported, however, where desirable. Deployment of a .NET assembly can be as simple as copying all of the files and contained sub-directories. This is referred to as XCopy deployment. Assembly deployment is compatible with Microsoft Installer (MSI) and by extension may be implemented using MSI compatible third-party installers.

'Side-by-side' is a methodology that brings needed support for incremental improvement of automation system software. It is also a methodology that allows fixing what is broken without breaking the things that aren't. What 'side-by-side' means is that multiple .NET software assemblies may be deployed together or one-after another, on a single computer, each depending upon different versions of a library or another software module. There is a unique mechanism called 'strong naming' that can be built into .NET assemblies and subsequently used to distinctly identify a code module along with its version. Strong naming utilizes a four part version number plus encrypted keys. This topic will be revisited in the section on security.

The mechanism insures that software modules will hook up to and run with the correct version of another module, even when differing versions are present on the system. Thus new assemblies may be deployed with changed library code while leaving the side-by-side existing modules unaffected.

For common modules that are used by many different programs, it is important to allow them to reside in a common place on a system where they will be easily found and thus shared. The Global Assembly Cache (GAC) in the CLR of each machine is that place. Assemblies that reside in the GAC must have 'strong names'. There is a standard mechanism, supported by MSI, for installing these widely used modules into the GAC.

Managed Code, Unmanaged Code and COM Interoperability

The CLR is a software system for running 'managed' code. Code that is built using Visual Studio.NET is 'managed' except for deliberate escapes into the realm of 'unmanaged' execution that are designated in the program source. Managed code utilizes a managed memory stack and a managed memory heap. Details about how the stack and heap work are quite involved, though not unfathomable. In addition, managed code must conform to a Common Type System (CTS). Interoperability between languages is assured because they incorporate the CTS and adhere to a Common Language Specification (CLS). CLS deals with the details for how data is represented and exchanged and how methods are invoked across module boundaries. CTS specifies data structure and rules as a base for an immense hierarchy of software object 'types'. Software objects built with .NET languages conform to Common Type Specifications. Instances of CTS objects are loaded onto CLR's heap. Simple data type variables and data structures are also represented as objects, however they are loaded onto CLR's stack. Understanding the distinction is important to for achieving optimal performance of managed code.

Managed code runs in a CLR environment called an 'application domain'. The CLR hosts assemblies within application domains and manages the interaction of the modules with system resources. Leveraging precise definitions of memory requirements and class methods (type safety) that are specified by the CTS, the CLR insures that the software running in the application domain 'behaves'.

The CLR also does one more important thing for managed code. It does 'garbage collection'. Garbage collection insures that memory and common software resources do not 'leak' as may be the case in unmanaged C++ code that is not carefully designed. Memory leaks have been a problem that plagues much of industrial control software. The often recommended repair for memory leaks with unmanaged C++ code is to reboot the computer, often called a 'sanity reboot'. That has been, or course, a drastic step. The .NET CLR's garbage collection mechanism for managed C#, VB.NET etc. effectively eliminates such leaks.

Questions can be raised however about predictability of managed code execution due to some uncertainty about when the garbage collection process actually runs. These questions will be treated in a later section of this paper.

.NET does support the C++ language compiled as 'unmanaged' code, i.e. translated into native machine code. Why would it be necessary to execute unmanaged code given the many stated advantages of .NET managed code? It turns out that there is a lot of well written, perfectly reliable C++ code out there. Such existing code isn't 'managed' C++.NET code, yet it does its job. The CLR permits interoperation between managed and unmanaged code, and in fact allows snippets of unmanaged code to be embedded directly in the managed code source files. Another reason for utilizing unmanaged code is that there are certain aspects of dealing with interface hardware, data communications and real-time execution that are not effectively achieved while complying with the rules of the managed CLR environment (in particular the asynchronous process that cleans up memory, namely garbage collection).

Microsoft's COM (Component Object Model) has engendered a huge installed base of industrial control software. The network extended version of COM is DCOM (D for distributed). An example using COM/DCOM for industrial purposes is OPC (OLE for Process Control). OPC is a worldwide, open standard for process data connectivity with process control systems, sensors, transmitters and intelligent devices. This large pool of well-tested, fully-functional code must be incorporated essentially as-is into a .NET environment. It would be too expensive, and the final delivery date would be way too late for commercial purposes if all industrial COM software were to suffer through the exercise of complete translation into the C# language.

The .NET languages provide for interoperability between managed CLR code and COM software. This is called COM Interop. There are actually two forms of COM Interop. For the first form C#, VB.NET or C++.NET managed code may be accessed as if they were COM components. This is done using a COM Callable Wrapper (CCW). Vice-versa, the managed code can leverage COM modules that are already built by referencing a Runtime Callable Wrapper (RCW) that encapsulates the COM code. In addition, C or C++ functions exported from DLLs can be called using the P/Invoke method call. It is very easy in Visual Studio.NET to import a COM component using the 'tlbimp.exe' tool. Once the COM component has been imported, or 'referenced' (terminology used in Visual Studio .NET) it is possible to browse the component's exposed interfaces and data members and reference it as if it were an intimate part of the entire .NET assembly. Bear in mind that these COM modules run as unmanaged code and if not carefully programmed they are subject to performance degrading memory leaks or unanticipated behavior excursions. Furthermore COM requires extreme care in registration with the operating system. They rely upon Globally Unique Identifiers (GUIDs) and don't actually provide 100% guarantee that they are what they say they are. Even though GUIDs are unique, the method of embedding a GUID into unmanaged COM components is itself not truly secure. Another issue is that DCOM requires error-prone registry configuration on both ends of a network connected system.

Security!

Industrial automation systems must be secure. They must be robust. They must be reliable.

Security for software involves several aspects. Every software module (e.g. a .NET assembly) must guarantee that it is the intended module for the automation strategy – no impostors, no out-of-date versions. Each module must play nicely within the confines of the computing

environment; meaning the module can't step out of its data structure memory, can't hog resources, can't hang threads, can't beat up other modules with incessant callbacks. Secure software modules must react to external method invocations only for authenticated users that are authorized to access the resources.

.NET provides support for detailed control over external interaction with managed software modules as well as fine-grained control over how each method deals with computing resources such as disk files, network protocol stacks etc. .NET is permeated by classes with methods that verify the security aspect of the execution of the code. In fact the most difficult thing for a programmer of industrial strength .NET CLR software may turn out to be properly leveraging these diverse security classes. Of significance is that Microsoft designed .NET to handle secure financial transactions distributed over Intranets and the Internet. These security mechanisms can be directly applied to industrial automation systems.(e.g. Web Services – to be covered later in this paper).

One of the most important robustness features of the .NET CLR is the implementation of the managed heap and stack with garbage collection. Furthermore the strong data typing of CTS protects against security breaches such as buffer overrun exploits.

Another common problem that can happen in software execution on an industrial automation controller is the execution of the wrong code. .NET assemblies with strong names can be deployed and monitored with a high level of confidence that they are executing the control code that is intended. Metadata included in an assembly (particularly the manifest) carries strong name information with encrypted security keys and an embedded hash code of the entire assembly. Taken together, this provides sufficient information for maintaining the integrity of code assemblies. There is however minimal out-of-the-box support for managing the deployment of a large number of individual assemblies. This can become an immense problem in a large, diverse automation system. The solution to this problem must be deferred to commercial, object oriented, industrial automation management systems which are being designed today.

Another aspect of industrial automation software regarding reliability is that in the face of increasing system entropy, they must continue to perform their designed functions flawlessly. One example of increased automation entropy is government mandated monitoring and controlled correction for effluents and emissions. No matter how much of this is mandated, the automated production control and safety systems must continue to work correctly for delivering economically viable products.

Significantly increased module count and the extreme complexity of interactions can lead to unpredictable system behavior. For a .NET framework system, it is possible to divide up the functionality of the system into classes of interdependent modules. CTS and CLS underpinnings of the CLR insure interoperability between modules. Even for communications transmitted over a network, the modules will understand each other. .NET includes a remote interoperability technology called .NET Remoting. Communication messages may be transmitted using open protocols such as SOAP (Simple Object Access Protocol) which is based upon XML (Extended Markup Language) or using binary protocols where high

performance is of concern. For added security, there are .NET support classes which make it straight forward to stream encrypted SOAP message payloads over networks, including the Internet. SOAP is a primary transport mechanism for the new Web Services paradigm. There is also support in the .NET framework for secured store-and-forward message exchange via MSMQ (Microsoft Message Queue). MSMQ implements its own brand of message delivery security.

Security to most people means the 'user name and password' dialogue box. It is true that authenticated sign-on provides an identity for a person that presumably will enter commands which an automation system must react to. Windows systems leverage identity and role-based security utilizing Domain Controllers that perform the service of authentication. Kerberos and other technologies borrowed from the UNIX world are now fundamental to Windows authentication. Symmetric and asymmetric public key encryption support is built into security classes of .NET. The .NET framework expands the concepts of security into the internals of software assemblies running in the CLR. There are two base-level security classes in the .NET framework, `WindowsPrincipal` and `WindowsIdentity` which are used programmatically to deal with security aspects of running software programs that access data and computing resources. These classes are applied in C#, Managed C++, VB.NET and J#.NET for securing access to runtime data fields and methods of .NET code. It is possible to limit interaction with .NET classes based upon authenticated roles impersonated by other software modules, whether on the same computer, over a LAN or over the Internet. This technology abstracts the concept of security such that multiple, distributed .NET assemblies can interact without human intervention over geographically dispersed areas while maintaining security. Again, it will be important in large industrial automation systems to have a system-wide management paradigm for planning, deploying and maintaining aspects of security for every deployed software object.

Configuration, ADO.NET, ASP.NET and Other Things That Thrive on XML

XML stands for Extended Markup Language. XML is built from SGML (Standard General Markup Language) and is a close cousin to ubiquitous HTML (Hypertext Markup Language). Some additional XML associated acronyms are: XSL (XML Stylesheet Language), XPath (an XML data query syntax), URI (Universal Resource Identifier), URL and URN (Universal Resource Locator and Universal Resource Name), DTD (Document Type Dictionary) and XML Schema (a syntax for defining how XML may be structured).

The most significant aspect of XML is that it is a syntax for representing structured information using ASCII or other encoding and it is human readable when decoded character by character to a computer display or printer. The .NET CLR relies heavily upon XML encoded configuration files through a class namespace called `System.Configuration`. What are 'namespaces'? They are a naming convention for identification in both XML Schema and most of the .NET languages supported by Visual Studio.NET. Namespaces are 'named' hierarchically in .NET by a sequence of strings separated by 'dots'. The 'root' of the namespace hierarchy of .NET framework classes is `System`. The namespace - `System.Configuration` - groups classes related to initialization and other externally configurable aspects of an assembly. In the C# language, declaring that 'Configuration' is to be a fundamental part of a code module is as simple as inserting the statement: `using System.Configuration.`

Namespaces in XML allow for short aliases that map to specific URIs in order to resolve namespace collisions between XML element tags. XML Schema namespaces are essential to validation of XML documents based upon industry oriented schemas. For example, there is a public OPC Foundation XML Schema that is represented as an XML namespace. The opportunity exists for the creation of other automation industry 'open' schemas with namespace designations. One advantage of using XML in automation software is that it affords clarity in configuration files thus improving development productivity and eliminating errors. There is now an opportunity for vendors of new industrial automation management systems to develop and publish for public review XML schemas covering configuration of standard industrial automation objects. Indeed this practice is already taking place for OPC Foundation and for Foundation Fieldbus.

Much code for the interpretation of and manipulation of XML came from open systems initiatives. Document parsers are an example of algorithmic code that originated and came to be refined through open system development. Parsers of XML documents do two things among others – check XML 'documents' as to whether they are well formed and validate those 'documents' against an XML Schema. Note that 'parsing' applies equally well to structured data streams that are exchanged between software assemblies. Microsoft originally built XML parsing into the IE (Internet Explorer) browser. They borrowed from open systems specifications. Now the .NET framework is permeated by XML with extensive classes for representing XML structured information, parsing it, and streaming it into and out of software application modules. System.Configuration is just one example of the use of XML in .NET. Taking advantage of XML in .NET affords productivity for automation software developers, in particular because there is extensive support in the framework for dealing with XML.

Other important sets of classes in .NET where XML plays a role are ADO.NET and ASP.NET. Instrument configuration databases, field bus device definition records, alarm logs, data historian records, equipment maintenance records, product lot tracking databases, shift reports – these are a small sample of information stores that utilize structured information that has something to do with industrial automation. ADO.NET encompasses data storage, data transmission and data processing technologies within the .NET framework. Though ADO.NET is open to the integration of any kind of database, out-of-the-box it supports Microsoft SQL and XML, including XML Schema. Integration of system configuration, data collection, data management and other data handling is closely tied to the choice of database technology. For any industrial automation system that chooses Microsoft SQL or its sibling MSDE (Microsoft Database Engine), the support provided through ADO.NET delivers huge productivity benefits for code development and testing. There are .NET framework classes for manipulating data based upon both SQL and XML. These are represented well in the C# language. For example, internal C# object structures can be derived simply by referencing an XML Schema. There are also simple yet powerful commands that stream entire datasets into and out of memory, with or without schema validation.

The benefit of leveraging ADO.NET for coding industrial automation objects extends beyond the stated advantage for the developer. In the world of rapidly evolving production systems there is an urgent need to 'rapidly' integrate with new software systems. For example, a facility may have added the capability to produce a new product with an elaborate chemical

composition. A new composition analyzer is then added for quality assurance. What if the new analyzer comes with its own computer? And that computer includes a database for storing product quality target profiles along with a history of analyzed values. If that database, out-of-the-box were based upon MSSQL or any SQL-compliant database, integration with the facility's automation system can be quickly developed and tested through the services provided by ADO.NET.

The web paradigm is arguably permeating the entire industrial workplace. Web pages containing business and personnel information are being delivered through Intranet and Internet Portal sites. The Web is becoming the de facto method for operating personnel to interact with such systems, and of course it is the preferred way now for industrial automation system vendors to deliver technical support. ASP.NET provides enhanced software tools for migrating the visualization of live industrial processes into the Web paradigm. Yet again, a major benefit of ASP.NET is developer productivity. Additionally ASP.NET promotes rapid deployment with secure, remote management of information.

The Web As An Operator Interface

There may be clamor of objection when the notion is raised that an operator could control his or her facility via a Web page. What needs to be recognized is that there are now technologies for leveraging secure, fast, reliable transactions over the Internet. The methods are being proven in the financial world where transactions carry an unmistakable economic penalty if they are not secure, fast, and reliable. Despite rampant reports about virus propagation, denial-of-service attacks, and theft identity, there are recognized technologies for dealing with each or these ailments.

Anti-virus protection, firewalls, intrusion detection, digitally-signed certificates, log-on authentication, and other techniques are based upon a variety of vendor proprietary and open technologies. Much of this technology originated in academia and has been thoroughly proven in UNIX as well as Windows. Microsoft has latched on to many of these public technologies and techniques, such as Kerberos security as mentioned above and digital certificates. An authentication technology developed by Microsoft that enables single sign-on is called Passport. Passport offers central management of user authentication even for the case where users switch to other web sites and perform tasks using differing applications. Passport now offers a development toolkit for a corporation to build its own Passport authentication service such that the corporation maintains full control. The .NET framework readily supports participation in systems where Passport authentication is used. In terms of industrial automation systems, the opportunity exists to provide secure, single sign-on support for mobile operators, technicians and supervisors thus simplifying the user interface and enabling faster navigation between applications.

ASP.NET provides a rapid application development paradigm with Web page development supported by Web Forms. For quick development and deployment, user interfaces via Web Forms can be used. This is particularly useful for configuration and presentation of any kind of database information. Managed code including C# may be used to build the database interfaces and link them to Web Forms. A lot of drag and drop technology is embedded in Visual Studio .NET for rapid creation of data oriented web applications.

COM+ For Business Integration

Another important technology is COM+. This was developed and released prior to .NET yet Microsoft has thoroughly integrated the development tools for .NET with support for COM+. The purpose of COM+ is to provide commercial strength capabilities to almost any code module that is built to consume COM+ services. These services effectively expand the reach of software originally built for a single user out to multiple users by incorporating data caching, user connection fan-out, session state management, etc. Enabling .NET assemblies such that their methods participate under COM+ services is genuinely as simple as placing one line of source code in front of a software method declaration. .NET does the rest under the covers.

Web Services - How To Deliver Ubiquitous Industrial Automation Services Anywhere, Anytime

There is a quiet revolution going on regarding technology for interconnecting software over the Internet. By the way, it is important to understand that Internet technologies can be effectively applied within the confines of an enterprise's Intranet. Getting back to the revolutionary aspect about interconnecting software, the buzz is about Web Services. The basic idea is open system integration for remotely connecting software to, and interoperating with, other software. Connecting software to other software over a network is not a new concept. Microsoft DCOM Java Remote Methods and CORBA (Common Object Request Broker Architecture) all support this. Each of these systems uses a closed binary wire protocol with custom marshalling. What Web Services brings to the table is a methodology for interconnection based upon an open method invocation syntax based upon – you may have guessed it – XML. SOAP (Simple Object Access Protocol) is a syntax for encoding remote method invocation calls between software applications. The connection can be HTTP or a binary format, both running over a UDP and/or TCP/IP Internet link.

There are two additional acronyms to introduce here that are fundamental to Web Services – WSDL (Web Services Description Language) and UDDI (Universal Definition and Discovery Interface). WSDL leverages XML and a Schema to specify in an unambiguous way what services are supported by a given software application and the data structures and data types for arguments when invoking the application's methods. UDDI represents an architecture for registering and discovering Web Services and their associated web methods..

What can Web Services do for industrial automation? The first answer is that they enable systems to access external software applications regardless of physical location or system platform. But are the connections secure? The same Web oriented authentication technologies defined above come into play. Why not use DCOM or Java Remote Methods? It is quite likely in a heterogeneous enterprise that the different ends of a connection will be built using different programming technologies and it is more likely that the internal structure of data will be quite different. Web Services addresses this issue by mandating an open specification of the required data structures and method syntax.

Finally, what does .NET bring to the table regarding productivity in the development of useable Web Services? During over four years of development of .NET, Microsoft has tightly integrated tools for building Web Services into Visual Studio.NET.

Citing an OPC example, there is a specification for serving up live, bi-directional data as a Web Service which is part of OPC XML-DA.

The Question of Industrial Automation Performance and Reliability

Vendors of industrial automation components by definition must deliver industrial strength software when embedded in those components. The components that don't work are scrapped and replaced by those from a competitor - that is if there is a truly functional competitive replacement. The test for such products is marketplace acceptance as refereed by organizations such as ISA, Gartner Group, and vertical market-specific standards organizations such as petroleum, pharmaceutical and automotive. The burden is upon the vendor to design a robust, high performance, reliable product, including the fact that any embedded software must function according to its specifications.

Testing, testing and more testing is needed even when given a solid design. Black box testing is not sufficient for most components because it usually only addresses carefully controlled scenarios. What is needed also is 'instrumentation' that monitors the performance of the component in the actual deployed context. The 'instrumentation' referred to here is software based. There are a lot of industrial automation components out there that run software which has essentially no software instrumentation to monitor them. Why is that? It is hard to add software instrumentation that monitors software in situ. When not done properly the software instrumentation either impedes performance or it causes unpredictable behavior in its own right.

Given the stated goal of having thoroughly tested industrial automation software with useful software instrumentation, how does .NET fit in?

First, the concept of 'realistic' abstraction may be postulated. Object modeling afforded in the C# language fosters the creation of real world oriented data structures and method argument signatures for 'realistic' classes. It is much easier to instrument (software-wise) realistic software objects. It becomes intuitively obvious what parameters need to be monitored because there are 'real' correlations.

Second, there are many .NET tools for profiling the performance of .NET CLR applications. The profiling can be built into the application. With careful design the profiling code will not significantly detract from performance. Windows Management Instrumentation (WMI) is the designation for a set of services and tools for attaching to software instrumented applications. WMI permits logging in intelligent ways. It permits secure user interfaces for tweaking configuration parameters of applications base upon observed performance data. It is even conceivable that WMI data can be folded back into an application such that the application can adjust its own performance, for example to level processor loading or even off-load some of its work to other applications, automatically. This technology may lead to the ultimate in self-tuning industrial control. The opportunity exists for entrepreneurial development of expert

'software instrumentation' leveraging the facilities of .NET profiling and WMI. Tack on Web Services and it is conceivable to arrive at a 'remote' control system optimization service. Such a service would leverage centralized domain expertise that tunes a client's control system according to current market economics for the customer's product and for asset utilization and maintenance purposes.

Doesn't Garbage Collection Make .NET CLR Unpredictable?

Much literature about the .NET CLR lauds garbage collection as a means to achieve reliable software. However that same literature warns of the fact that a programmer can't rely upon scheduling the exact times when garbage collection occurs. Certainly the CLR must take time away from other software processes to clean up a 'dirty' managed heap and tidy up the managed stack. Certainly for real-time industrial control systems there are cases where even a slight delay or small variation in the 'rhythm' of collection events will interfere with true real-time control.

There is not a simple answer. It can't be said unilaterally that garbage collection can be absolutely controlled by the programmer. There is some degree of control through methods that tell the CLR to try to collect garbage immediately, however as pointed out in programming documentation on .NET, even immediate calls for garbage collection do not result in an immediate return and resumption of code processing.

A quick fix is elusive. One approach is to minimize the garbage generated in the first place through careful allocation and control of object instances. It is also possible to relegate to 'unmanaged' code key object management that needs absolute control over software object lifetimes. In any event this is another of those opportunities for innovation. Be sure to remember the suggestion about testing, testing, more testing, and slapping on software instrumentation for good measure.

.NET on Embedded Windows XP and CE.NET

There persists a gut level reluctance regarding deployment of critical industrial control system software on Windows, even when it is Windows XP. Windows XP has a lot going for it in robustness but it doesn't yet have the playing time to afford it veteran status. Probably the biggest issue for any of the Windows operating systems is that there is a lot of seemingly unnecessary software loaded up and lurking in the background on an 'office-class' computer. This seems particularly true of the services built in that support stuff like Microsoft Office.

Embedded XP is a version of Microsoft's new operating system that provides the opportunity to build a leaner version that doesn't carry all of the baggage. By eliminating the stuff that is unusable in an industrial controller, there will be better performance and higher reliability, not to mention the fact that a smaller hard disk, or even flash disk is possible. Furthermore the memory footprint can be reduced. Eliminating the extra stuff allows for inclusion of software instrumentation as long as the facilities of WMI are left intact in the Embedded XP version. The .NET Framework will indeed run on Embedded XP thus allowing .NET applications to execute within a controlled, sealed environment.

CE.NET is targeted at the Windows CE environment which can be installed to run on a number of low-cost, small footprint CPUs. The Compact CLR is the version designed to run on CE. There is a reduced set of framework classes such that not all .NET code will transfer unchanged to CE.NET. However the opposite will generally be true. Any code that runs on CE.NET will be able to be run on Windows XP and Windows Embedded XP. The real issues for this interoperability will be the installed I/O hardware. Kernel level drivers will most likely be different between XP and CE.

.NET Spans Up and Down Levels of Control and Integration

Productivity in development of enterprise industrial control and integration software relies heavily upon the skill sets of individual programmers. A variation of the 'hammer and nail' adage is appropriate. Every level of software integration in an enterprise has its proprietary toolset. Integration with a SAP ERP system, for example, requires ABAP/4 along with very specific support code modules. Oracle databases for inventory management use Oracle tools. Getting down to the nitty-gritty of building a PID controller or a unit sequencer, a lot of code is written in C++ or even C with the target being a proprietary 'real-time' operating system (RTOS).

An RTOS will still be required where fine-grained millisecond by millisecond control over external communications is needed. But those tasks are being pushed further out into the field and can generally be embedded in the field device itself. Fieldbus, Profibus, et al can be used to link to those devices and an OPC I/O server can round up all of the data from those devices.

.NET can be programmed to achieve many, even most, of the industrial control functionality above that layer of field-installed real time. What is more important is that .NET is ideal for building integration software that ties together enterprise software applications and systems. The extensive support in .NET for COM+ provides scalability leveraging reusable .NET assemblies with connection fan-out, resource pooling, transaction management and other enterprise level functions. In particular the framework software provided with .NET such as languages and classes in Visual Studio.NET deliver a new kind of functionality similar to that of a multi-purpose tool. It can be likened to a Swiss Army knife with screwdriver, blade, corkscrew and even a toothpick (though there isn't really a hammer in there). Since the problem set of most enterprise integration scenarios doesn't come in the form of nails, a hammer won't be the appropriate tool. It is highly probable, however, that a tool in our ".NET Swiss Army knife" will get the job done. Of course there will be situations when a specialty tool will be required, but .NET is capable of stretching the 80-20 rule to 90-10, i.e. providing functional solutions for up to 90% of real world problem sets.

Many integration problems involving the connection of business systems to industrial control systems currently exist. These must be solved in order to achieve maximum manufacturing efficiency. A major roadblock exists because these tasks must be performed by a very limited pool of skilled talent. What this talent pool needs are ubiquitous industrial automation tools along with software applications, software objects, and software instrumentation. It will be up to enterprising vendors and agile entrepreneurs to deliver these. The premise of this paper is that .NET can be leveraged by skilled programmers to achieve high levels of productivity during the

implementation of manufacturing automation solutions up and down all levels of control and integration within and between enterprises.

Summary

Actually the last statement in the previous paragraph provides the summary in a nutshell. Amplifying the skills of industrial control programmers is what is needed to deliver a lot of functionality in a uniform way while insuring interoperability. .NET CLR, C#, Web Services and Embedded XP, along with many other technologies, will help improve productivity for the development of industrial automation systems moving forward.

Glossary

.NET - a Microsoft networking and computing framework

.NET Assembly - the set of files grouped together in support of a .NET application, .NET service or .NET DLL

.NET Remoting – a mechanism for extending .NET Assembly method calls over a network

ABAP - Application Business Access Protocols, a high level language for SAP's ERP system

ADO.NET - ActiveX Database Objects for .NET, a database connection, query and update methodology principally used with Microsoft SQL Server

Application domain - a .NET CLR program execution model that insures type safety, proper threading plus resource and memory sharing

ASP.NET - ActiveX Server Pages for .NET, a framework supporting multiple languages attached to an IIS (Internet Information Server) Web Server

C# - an object oriented programming language created for the .NET framework

CCW - COM Callable Wrapper, classes in .NET languages that 'wrap' its code such that it appears to be a COM module

CLR - Common Language Runtime, the runtime environment for executing .NET applications

CPU - central processing unit

C++.NET - C++ object oriented programming language for the .NET framework which is a superset of ANSI Standard C++

CLI - Common Language Infrastructure, specification of language rules and structures for the .NET CLR

CLS - Common Language Specification, definition of syntax and interface methods insuring interoperability between .NET languages

COM - Component Object Model - a code structure and interface specification supporting module interacting through interfaces

COM Interop - the method whereby .NET languages are interfaced with COM based code

COM+ - COM with services that support scalability, resource pooling and transactions

CORBA - Common Object Request Broker Architecture, a specification for software object method interfaces created by the OMG (Object Management Group)

CTS - Common Type System, a specification of the data structures and interface methods for common object types in .NET

DCOM - Distributed Component Object Model, COM connected via a network (see COM)

DLL - Dynamic Link Library

DTD - Document Type Dictionary, a structured specification of how to organize an XML document, archaic, but still in use (see XML Schema)

Entropy - a concept from the science of physics that describes the state of complexity and/or confusion of a large collection of interacting physical entities

ERP - Enterprise Resource Planning

Fieldbus - a data communications bus for field devices

GAC - Global Assembly Cache, a directory structure and the runtime code management for shared .NET Assemblies

Garbage collection - the mechanism in .NET CLR that cleans up stack and heap resources

GUID - Globally Unique Identifier - an encoded string that is guaranteed to be unique across the universe deployed modules

Hash code - a number or unique string that is generated to facilitate searching into a dataset or memory structure

HTML - Hypertext Markup Language, a specification base upon SGML used for encoding Web pages (see SGML)

HTTP - Hypertext Transport Protocol, a protocol for moving data over the Internet

IL - Intermediate Language (see MSIL)

IP - Inter-network Protocol, an inter-network data exchange protocol

Java Byte Code - a common 'source' language interpreted in a Java Virtual Machine which is a runtime environment

Java Remote Methods (JRM) - a specification for how Java applications interoperate over a network and the Internet

JIT - Just-In-Time, the mode of the .NET CLR whereby IL code is compiled to byte code for execution on the machine

JScript.NET - a Java-like program scripting language for Web pages in the .NET framework

J#.NET - a Java-like language designed for the .NET framework

Kerberos - a specification for security encoding and authentication methods for networks and the Internet

LAN - Local Area Network

Managed code - software code that executes in the .NET CLR environment

Manifest - the metadata contained in a .NET program that thoroughly describes the assemblies, their content, required DLLs, etc.

MFC - Microsoft Foundation Classes - a hierarchical set of code modules supporting Win32 on Windows operating systems

MSDE - Microsoft Database Engine, a compact version of Microsoft SQL Server

MSI - Microsoft Installer, a specification and support software for deploying and installing Windows applications

MSIL - Microsoft Intermediate Language, a common 'source' language used for runtime .NET CLR which is JIT-compiled into native code for machine execution

MSMQ - Microsoft Message Queue, a data transmission technology for networks with secure, guaranteed delivery involving store and forward

MSSQL - Microsoft SQL Server, a database engine supporting SQL (Structured Query Language) now also supporting XML

Namespace - in .NET a namespace is a group of related object classes in a hierarchy; for XML namespace identifies an XML Schema reference for encoding

OPC - OLE (Object Linking and Embedding) for Process Control, a specification and tools for connecting industrial devices

Passport - a Microsoft system for user authentication that extends across multiple, distributed Web enabled applications

P/Invoke - a call in C# language that 'invokes' a method of a wrapped COM module

Profibus - a data communications bus for field devices

RCW - Runtime Callable Wrapper, classes that 'wrap' COM modules such that they can be called from within .NET languages

RTOS - Real-time Operating System

Side-by-Side - a code deployment methodology that insures execution of .NET Assemblies utilizing correct module versions

SGML - Standard General Markup Language, a specification for syntax used in the creation of structured information languages for the Internet

SOAP - Simple Object Access Protocol, used to encode information using XML for invoking software program methods over the Internet

Strong naming - a methodology for embedding secure unique identity information in a .NET Assembly

TCP - Transport Control Protocol, a network data exchange control protocol

UDDI - Universal Definition and Discovery Interface

UDP - User Datagram Protocol, a network data exchange protocol

Unmanaged code - any software code that executes outside of the .NET CLR environment

URI - Universal Resource Identifier, a string intended to uniquely identify a document or object associated with the Internet

URL - Universal Resource Locator, a string utilized by the Web to access a web server or other web resource

URN - Universal Resource Name, a string intended to identify a document or object in a program context associated with Internet access

VB.NET - Visual Basic programming language for the .NET framework

Visual Studio .NET, a programming integrated development environment

Web Service - a software module that exposes methods that may be invoked over an Internet or Intranet connection

WMI - Windows Management Instrumentation, software instrumentation technology for monitoring applications and computing resources

WSDL - Web Services Description Language

XCopy - a command line initiated executable that copies files and file directories en masse from place to place

XML - Extended Markup Language, a syntax for organizing information in a structured way with human-readable character encoding (derived from SGML)

XML-DA - an OPC specification for industrial data exchange utilizing XML encoding

XML Schema - an XML document that is structured as a specification for how to create an XML document (replaces DTDs in this role)

XPath - XML Path, a specification for utilizing XML to identify search criteria into an XML document

XSL - XML Stylesheet Language, a syntax for specifying how XML documents can be transformed to HTML or different XML

References

Burton, Kevin, .NET Common Language Runtime Unleashed, Sams Publishing, ISBN 0-672-32124-6

Troelsen, Andrew, C# and the .NET Platform, Apress, ISBN 1-893115-59-3

The following articles are found in issues of The Microsoft Journal for Developers MSDN Magazine, publisher CMP Media LLC:

- Richter, Jeffrey, "Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web", September 2000 MSDN issue
- Kirtland, Mary, "The Programmable Web, Web Services Provide Building Blocks for the Microsoft .NET Framework", September 2000 issue
- Pietrek, Matt, "Avoiding DLL Hell, Introducing Application Metadata in the Microsoft .NET Framework", October 2000 MSDN issue
- Richter, Jeffrey, "Part 2, Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web", October 2000 MSDN issue
- Richter, Jeffrey, "Garbage Collection, Automatic Memory Management in the Microsoft .NET Framework", December 2000 MSDN issue
- Richter, Jeffrey, "Garbage Collection, Part 2, Automatic Memory Management in the Microsoft .NET Framework", December 2000 MSDN issue
- Box, Don, "House of COM, Is COM Dead?", December 2000 MSDN issue
- Richter, Jeffrey, ".NET, Type Fundamentals", December 2000 MSDN issue
- Skonnard, Aaron, "XML in .NET, .NET Framework XML Classes and C# Offer Simple, Scalable Data Manipulation", January 2001 MSDN issue
- Richter, Jeffrey, ".NET Framework, Building, Packaging, Deploying, and Administering Applications and Types", February 2001 MSDN issue
- Brown, Keith, "Security in .NET, Enforce Code Access Rights with the Common Language Runtime", February 2001 MSDN issue
- Olson, Lance, ".NET P2P, Writing Peer-to-Peer Networked Apps with the Microsoft .NET Framework", February 2001 MSDN issue
- Prattschner, Steven, "Microsoft .NET, Implement a Custom Common Language Runtime Host for Your Managed App", March 2001 MSDN issue
- Richter, Jeffrey, ".NET Framework, Building, Packaging, Deploying, and Administering Applications and Types, Part 2", March 2001 MSDN issue
- Richter, Jeffrey, ".NET, An Introduction to Delegates", April 2001 MSDN issue
- Richter, Jeffrey, ".NET Delegates Part 2", June 2001 MSDN issue
- Sells, Chris, "Visual Studio .NET, Managed Extensions Bring .NET CLR Support to C++", July 2001 MSDN issue
- Platt, David S. "Microsoft .NET Interop, Get Ready for Microsoft .NET by Using Wrappers to Interact with COM-based Apps", August 2001 MSDN issue
- Richter, Jeffrey, ".NET, Implementation of Events and Delegates", August 2001 MSDN issue
- Ewald, Tim, "COM+ Integration, How .NET Enterprise Services Can Help You Build Distributed Applications", October 2001 MSDN issue
- Richter, Jeffrey, ".NET, Enumerated Types", October 2001 MSDN issue

Kasiolas, Anastasios, “.NET CLR Profiling Services, Track Your Managed Components to Boost Application Performance”, November 2001 MSDN issue

Getz, Ken, “Windows Services, New Base Classes in .NET Make Writing a Windows Service Easy”, December 2001 MSDN issue

Pietrek, Matt, “The .NET Profiling API and the DNProfiler Tool,”, December 2001 MSDN issue

Restrepo, Tomas, “Visual C++.NET, Tips and Tricks to Bolster Your Managed C++ Code in Visual Studio .NET”, February 2002 MSDN issue

Lippman, Stanley B., “Still in Love with C++, Modern Language Features Enhance the Visual C++ .NET Compiler”, February 2002 MSDN issue

Richter, Jeffery, “.NET, Array Types in .NET”, February 2002 MSDN issue

Richter, Jeffrey, “.NET, Run-time Serialization”, April 2002 MSDN issue

Lowy, Juval, “Security, Unify the Role-based Security Models for Enterprise and Application Domains with .NET”, May 2002 MSDN issue

Pozen, Zina, “WMI and .NET, System.Management Lets You Take Advantage of WMI APIs within Managed Code”, May 2002 MSDN issue

Fox, Dan, “Security, Protect Private Data in Windows with the Cryptography Namespace of the .NET Framework”, June 2002 MSDN issue

Schafer, J. Andrew, “C#, XML Comments Let You Build Documentation Directly From Your Visual Studio .NET Source Files”, June 2002 MSDN issue

Whittington, Jason, “Rotor, Shared Source CLI Provides Source Code for a FreeBSD Implementation of .NET”, July 2002 MSDN issue

Yao, Paul, “Windows CE.NET, New Version Offers Revamped Platform Builder, Improved Tools, Enhanced API, and Source Code”, July 2002 MSDN issue

Sells, Chris, “.NET Zero Deployment, Security and Version Models in the Windows Forms Engine Help You Create and Deploy Smart Clients”, July 2002 MSDN issue

Richter, Jeffrey, “.NET, Run-time Serialization Part 2”, July 2002 MSDN issue

Box, Don, “Security in .NET, The Security Infrastructure of the CLR Provides Evidence, Policy, Permissions, and Enforcement Services”, September 2002 MSDN issue

Microsoft, Windows, Jscript, Visual J#, MSDN, Visual Basic, Visual C++, Visual C#, Visual Studio, and Win32 are registered trademarks of Microsoft Corporation. Other trademarks or trade names are mentioned herein are the property of their respective owners.